Initiation à linux/UNIX et au langage ForTran

Table des matières

1 UNIX et linux **2**

1.1 L'arborescence UNIX et linux . . . 2

1.2 Commandes de base . . . 2

2 ForTran **2**

2.1 Introduction . . . 3

2.1.1 Informations et recommandations importantes . . . 3

2.1.2 Compilation et exécution . . . 4

2.2 Variables, paramètres et types . . . 4

2.3 Structure d'un programme . . . 4

2.4 Expressions et opérateurs logiques . . . 5

2.5 Fonctions prédéfinies . . . 5

2.6 Instructions de contrôle (if et do) . . . 5

2.6.1 Les branchements conditionnels . . . 6

2.6.2 Les boucles . . . 6

2.7 Accès fichiers . . . 7

2.8 Sous-programmes . . . 7

2.8.1 Fonction . . . 7

2.8.2 Sous-routine . . . 7

3 Exercices **8**

3.1 Bissextile . . . 8

3.2 Somme et moyenne . . . 8

3.3 Tri et calcul de médiane . . . 8

3.4 Transformée de Fourier . . . 9

1 UNIX et linux

1.1 L'arborescence UNIX et linux

UNIX et linux sont des systèmes d'exploitation qui comportent, en général, plusieurs répertoires (liés ou non aux partitions ou disques). Quelques-uns, parmi les plus classiques et auxquels vous ne toucherez pas, sont :
`/` : c'est la racine, le début de l'adresse absolue (le chemin pour accéder à un répertoire ou un fichier commencera toujours par /).
`/sbin` : contient les exéxcusables indispensables pour le bon fonctionnement du système.
`/dev` : contient les points d'entrée des périphériques (*devices* en anglais).
`/boot` : contient les fichiers indispensables pour démarrer le système.
`/root` : contient les fichiers de l'administrateur (celui qui a le droit de modifier le système).
`/home` : contient les fichiers personnels des différents utilisateurs par défaut.
Sur des serveurs multi-utilisateurs, il est possible que le répertoire par défaut ne soit pas `/home/nom`. Pour connaître le chemin exact du répertoire dans lequel vous êtes autorisé à écrire (et qu'on appellera ensuite le *home*), il suffit d'ouvrir une fenêtre *terminal* et de taper :

```
cd
pwd
```

La première commande (change directory) vous ramène dans votre *home*, la seconde (print working directory) donne comme réponse le chemin d'accès. En général, c'est */home/login*. Si vous naviguez dans vos répertoires et que vous ne savez plus où vous êtes, vous pouvez taper cette commande (pwd) à tout moment.

Note importante : Sous UNIX et linux, les minuscules et les majuscules ont leur importance : mkdir est une commande différente de mKdir, les répertoires toto et Toto sont différents.

1.2 Commandes de base

`cd` : change directory. Exemple : `cd Toto` permet d'aller dans le répertoire Toto (s'il est contenu dans le répertoire dans lequel vous êtes).
`ls` : list. Donne la liste de tout ce qu'il y a dans le réperoire dans lequel vous êtes (fichiers et répertoires).
`mkdir` : make directory. Crée un répertoire. Exemple : `mkdir Toto` permet de créer le répertoire Toto.
`rm` : remove. Permet d'effacer un fichier (`rm nom-fichier`). Pour effacer un répertoire, il faut utiliser `rmdir nom-répertoire` mais celui-ci doit être vide.
`cp` : copy. Permet de copier un fichier. Exemple : `cp fich1.txt Toto/fich2.txt` recopie le fichier fich1.txt dans le sous-répertoire Toto en lui donnant le nom de fich2.txt.
`xemacs` : permet de lancer cet éditeur de texte.
`man` : manuel. Permet d'accéder au manuel d'utilisation des commandes. Exemple : `man mkdir` explique ce que fait la commande mkdir et comment on l'utilise.

2 ForTran

Plusieurs livres sur le langage ForTran pouvant vous être utile sont disponibles à la BU : Delannoy (1993) et Lignelet (1996), par exemple. Le manuel de Cambridge (*Computational Physics - Self study*

guide 2 - Programming in Fortran 95) qui vous est fourni est une très bonne base pour débuter. Vous avez dedans beaucoup d'exemples et de petits exercices corrigés qui illustrent ce qui est présenté dans les paragraphes ci-dessous.

2.1 Introduction

Le ForTran est un langage de programmation dont la première version a été écrite en 1955 par IBM (il signifie *FORmula TRANslating*). Il est donc principalement dédié à la formulation mathématique et se rencontre beaucoup dans les milieux universitaires et de recherche en général. En 1962, est apparu le premier compilateur ForTran IV mais les grandes modifications sont apparues en 1977 (avec le ForTran 77 ou f77). Depuis, deux autres versions de ForTran ont été développées (f90 et f95), permettant notamment la déclaration de types structurés, l'allocation dynamique de la mémoire, l'utilisation des pointeurs et la construction modulaire.

2.1.1 Informations et recommandations importantes

- Dans un programme, les commentaires sont indispensables, en ForTran comme dans tout langage de programmation. Cela permet d'échanger ses codes, de travailler à plusieurs et de pouvoir comprendre plusieurs années après le travail fait par quelqu'un (y compris soi-même) ; en ForTran, une ligne de commentaires commence toujours par « ! ».
- Le langage ForTran ne fait pas la différence entre les minuscules et les majuscules, la variable `sop` est la même que `SoP`.
- Un programme ForTran devrait toujours être commencer avec la commande `implicit none`. Celle-ci indique qu'aucune variable n'a de type (entier, réel, chaîne de caractères etc.) a priori. Il faut donc toutes les définir.
- Il existe deux manières d'afficher du texte ou des valeurs à l'écran, `write(*,*)` et `print*,`. La seconde est très utile pour trouver les erreurs dans un programme (débugger). Les fonctions `read` et `write`, s'utilisent aussi bien pour les lectures au clavier (par exemple quand le programme demande une valeur à l'utilisateur lors de son exécution) et l'écriture à l'écran qu'avec des fichiers ASCII, par exemple pour utiliser un fichier contenant des données.
- L'affectation est le fait de donner une valeur à une variable. Par exemple, si la valeur 2.0 doit être affectée à la variable `x`, on écrira
  ```
  x = 2.0
  ```
  Attention, cela n'a pas de lien avec le signe = au sens mathématique, car si on veut augmenter la valeur de `x` de 1.5, on écrira :
  ```
  x = x+1.5
  ```
- Un tableau (un ensemble de variables rangées dans un ordre donné) se remplit, s'utilise et se parcourt toujours à l'aide d'une boucle `do` (voir manuel de Cambridge), ou plusieurs si le tableau est à plusieurs dimensions.
- Un programme s'écrit en recherchant toujours le plus haut niveau d'abstraction ; dans ce sens, il est préférable de ne pas fixer de valeurs *en dur* dans le code. Par exemple, si vous devez lire des données dans un fichier qui contient 25 lignes et que vous fixez ce nombre de lignes dans votre programme, celui-ci ne marchera plus le jour où on vous donnera un fichier de 30 lignes, et ceci même si ce que doit faire le programme avec ces données ne change pas.
- La compilation (et souvent l'exécution) est une action qui normalement prend à peine quelques secondes, compiler toutes les 10 lignes de codes est une garantie de vite localiser les erreurs et les *bugs*.

2.1.2 Compilation et exécution

En général, le compilateur s'appelle `f95` ou `g95`.
Exemple : le programme à compiler s'appelle `prog.f95`. Il faut alors taper `f95 prog.f95 -o prog.out`. L'exécutable est alors `prog.out` que l'on exécute en tapant `./prog.out`.
La compilation peut se faire en plusieurs étapes (création des objets `.o`). Il est aussi possible d'écrire un script qui organise la compilation en plusieurs étapes, pour ne pas avoir à retaper tout le temps plusieurs commandes, ou d'écrire un *makefile*.

2.2 Variables, paramètres et types

Il est important de faire la différence entre paramètres et variables. Au sens premier du terme, les deux sont des variables mais les paramètres ne sont normalement pas destinés à changer de valeur au cours de l'exécution du programme. Ceux-ci peuvent s'écrire soit dans le programme principal, avant la déclaration des variables, soit dans un module externe.

Variables et paramètres sont typés, c'est-à-dire qu'ils appartiennent à une famille de variables. La syntaxe est la même pour tous les types de variables :

```
type :: variable
```

S'il y a plusieurs variables d'un même type, elles peuvent s'écrire à la suite, séparées par des virgules.
Les types plus classiques sont :

- character
- integer
- logical
- real
- complex
- real double precision

Mis à part les caractères - qui sont déjà des chaînes de caractères donc des tableaux - les tableaux peuvent être de tous les types. Ils se déclarent avec la commande `dimension`. Ils peuvent être de plusieurs dimensions (lignes, colonnes et plus) et le nombre de cases est soit fixé par un paramètre, soit allouable dynamiquement avec la commande `allocate`.

2.3 Structure d'un programme

Le langage f95 peut compiler plusieurs sous programmes pour obtenir des objets (`.o`), qui seront ensuite liés entre eux pour créer l'exécutable. Le programme principal peut donc utiliser des modules et d'autres sous-programmes, écrits, par exemple, par d'autres personnes. Les modules servent à définir soit des constantes, soit des fonctions génériques. Il s'écrit dans un fichier à part :

```
module gprms
! module pour définir les paramètres globaux !
   implicit none
   integer, parameter :: nn = 12
   integer, parameter :: sp = kind(1.0)
   integer, parameter :: dp = kind(1.0d0)
end module gprms
```

Les paramètres `sp` et `dp` pourront être par la suite utilisés dans un programme pour définir des réels simple (7 chiffres significatifs) ou double précision (15 chiffres significatifs), voir détails dans le chapitre 7 du manuel de Cambridge. Les modules peuvent être utilisés par plusieurs programmes.
Voici une structure classique et ordonnée pour un programme f95 :

```
program example
   use gprms ! permet d'utiliser les paramètres déclarés dans le module
   implicit none
! Déclarations des variables
   integer :: i,nb
   real(sp), dimension(nn) :: tab  ! tableau de 12 réels simple précision

! Initialisation des variables (mise à 0 des valeurs du tableau)
   do i=1,imax
      tab(i)=0.0
   end do

! Corps du programme (liste des instructions)

end program example
```

2.4 Expressions et opérateurs logiques

Les expressions logiques sont des expressions booléennes, elles ne peuvent prendre que deux valeurs : vrai ou faux. Ex : Est-ce que i est plus grand que j ?
Pour cela il est nécessaire d'avoir des opérateurs de comparaison :

`>`	et	`>=`
`<`	et	`<=`
`==`	et	`/=`

Pour les expressions plus complexes, il existe les opérateur logiques : `.and.`, `.or.` et `.not.`
Ex : `(i < j).and.(i /= 0)`

2.5 Fonctions prédéfinies

ForTran est un langage qui possède beaucoup de fonctions prédéfinies (ou intrinsèques). Les noms des fonctions sont réservées, si une variable, un paramètre ou une routine est déclarée avec un nom de fonction prédéfinie, la compilation ne pourra pas aboutir. Parmi celles-ci les fonctions numériques sont les plus utiles.

- fonctions élémentaires de conversion : `abs`, `aimag`, `aint`, `cmplx`, `real`, ...
- fonctions élémentaires sans conversion : `mod`, `sign`, ...
- fonctions élémentaires mathématiques : `acos`, `asin`, `cos`, `cosh`, `sin`, `sinh`, `atan`, `tan`, `exp`, `log`, `log10`, `sqrt`, ...

La valeur de π est à définir, on pourra prendre par exemple, `pi = 4.0*atan(1.0)`.

2.6 Instructions de contrôle (if et do)

Ce sont les deux grandes notions de la programmation, elle suffisent, dans 99,9% des cas, à aborder toutes les situations.

2.6.1 Les branchements conditionnels

Le branchement conditionnel le plus simple est : `if`.

```
if (expression logique) then
   conséquence du if
else ! facultatif
   conséquence du else
end if
```

Il peut également s'utliser sans le `else`. De plus, si un autre choix peut se présenter avec le `else`, il est possible d'utiliser la fonction `elseif`.
Enfin, pour un choix multiple, la commande `select case` sera plus utile.

```
...
select case(i)
   case (0) ! cas où i=0
      ...
   case (1,2) ! cas où i vaut 1 ou 2
...
   case (5:10) ! cas où i est compris entre 5 et 10
      ...
   case (11:) ! cas où i est plus grand que 11
...
   case default ! tous les autres cas
      ...
end select
```

2.6.2 Les boucles

Trois boucles possibles :

1. le nombre d'itérations est connu :

```
...
n = 37
...
do i = 1,n
   tablo(i) = 4.0*real(i)**2+2.3
end do
...
```

2. les itérations sont valables tant que :

```
...
j = 37
...
i = 1
do while (i <= j)
   tablo(i) = 4.0*real(i)**2+2.3
   i = i+1
end do
...
```

3. le critère de sortie de boucle est connu :

```
...
j = 37
...
i = 0
do
   i = i+1
   tablo(i) = 4.0*real(i)**2+2.3
   if (i == j) exit
end do
```

Le compteur (ou indice) est forcément de type entier et est une variable muette. Il ne doit **surtout pas être modifié** dans le corps de la boucle.

2.7 Accès fichiers

Les accès en lecture ou en écriture suivent la même logique : ils doivent être le plus rapide possible. Le fichier est assigné à une unité (un nombre) au moment de l'ouverture. Cette unité sera ensuite utilisée partout (à la place du nom du fichier).

- Ouverture : `open`
- Lire : `read`
- Écrire : `write`
- Fermer : `close`

Voir des exemples dans le manuel de Cambridge.

2.8 Sous-programmes

Les deux types de sous-programmes s'écrivent exactement comme un programme principal. Suivant les besoins, ils peuvent être internes, c'est-à-dire écrits à la suite du programme principal, soit externes. En général, si le rôle du sous-programme est très global, il sera externe (ex : compte le nombre de lettres dans une chaîne de caractères). En f95, la différence entre fonction et sous-routine n'est plus tout à fait évidente, les deux peuvent être souvent interchangées.

2.8.1 Fonction

La fonction est à relier au sens de la fonction mathématique. Elle renvoi une valeur qui porte le nom de la fonction. Petit inconvénient, elle ne retourne qu'une valeur ; elle peut toutefois agir sur une variable globale (définie dans un module, par exemple).
Elle débute par le mot clé `function` et se termine par `end function`.

2.8.2 Sous-routine

La sous-routine est plus générale. Elle peut renvoyer une valeur, agir sur une variable globale et faire des actions. C'est typiquement des petits bouts du programme principal qu'il est plus judicieux de placer ailleurs, ex : lire un fichier, écrire un fichier, faire un tri dans un tableau.
Elle débute par le mot clé `subroutine` et se termine par `end subroutine`.

3 Exercices

3.1 Bissextile

Un chiffre est pair s'il est divisible par 2. Pour une année bissextile, les règles sont les suivantes :
- soit divisible par 4 mais non divisible par 100,
- soit divisible par 400.

La proposition contraire stipule qu'une année n'est pas bissextile si elle est
- soit non divisible par 4,
- soit divisible par 100, mais pas par 400.

2009 n'est pas bissextile. 2008 était bissextile (car divisible par 4). L'an 1900 n'était pas bissextile, car bien divisible par 100 mais non divisible par 400. L'an 2000 était bissextile car divisible par 400.

Écrire un programme, appelé « `bissextile.f95` », permettant à l'utilisateur de savoir si une année (chiffre tapé au clavier) est paire ou impaire et si elle est ou non bissextile.

3.2 Somme et moyenne

On considère 11 chiffres

position	1	2	3	4	5	6	7	8	9	10	11
valeur	-2	3	34	7	$\sin(\frac{\pi}{5})$	2,3	-9.3	$\sqrt{(3)}$	23	47.2	9

1. Écrire un programme, dans lequel l'affectation des valeurs dans le tableau sera écrite. Afficher le résultat à l'écran.
2. Écrire la partie de programme permettant de calculer la somme et la moyenne des 11 valeurs. Afficher le résultat à l'écran.
3. Mettre la partie calcul dans un sous-programme, appelé `sommoy` ; la partie affichage final doit rester dans le programme principal.
4. Écrire les 11 valeurs dans un fichier ASCII (ou texte) et écrire la routine permettant de lire ces valeurs depuis le programme, avant de faire les calculs codés précédemment.

3.3 Tri et calcul de médiane

Définition : le terme *médiane* vient du latin *medius*, qui signifie au milieu. La médiane est la valeur qui permet de partager une suite de valeurs ordonnées en deux parties égales en termes de nombre d'éléments. Pour la calculer, il faut donc ranger les valeurs par ordre croissant et trouver celle qui est au milieu. Cela est immédiat pour un nombre impair (puisque égal à $2n + 1$) et pour le cas d'un nombre pair, la valeur retenue est la moyenne des deux valeurs médianes.
Ex : soit la série $5, 2, 9, 4, 3$; une fois rangée, on obtient la suite ordonnée $2, 3, \mathbf{4}, 5, 9$; la médiane est immédiatement la valeur 4.
Si maintenant la première valeur est supprimée, la suite devient $2, 9, 4, 3$ et une fois rangée : $2, 3, 4, 9$; la médiane est alors $\frac{1}{2}(3 + 4) = 3, 5$.

1. On s'intéresse au salaire médian d'une population de onze personnes. Sur ces onze personnes, dix gagne le Salaire Minimum Interprofessionnel Garanti (SMIG) et le onzième gagne dix fois le SMIG. Écrire un programme qui permette de calculer le salaire moyen et le salaire médian. L'idée que l'on se fait du mot « salaire moyen » est-il plus proche de la notion de moyenne ou de médiane ?

2. Dans le fichier `data1.txt` à prendre sur la page web d'Éric Beucler, figurent 235 valeurs. Ces valeurs sont visualisables, par exemple grâce au logiciel `xmgrace` :
   `>> xmgrace data1.txt`
3. Ces valeurs sont rangées dans un ordre aléatoire sur une seule colonne, étant supposé que le numéro de la ligne dans le fichier correspond au numéro de la valeur. Écrire un autre programme permettant de lire ces valeurs, puis de les trier par ordre croissant, puis d'en calculer la médiane. Calculer également la moyenne, commenter.
4. Tester que l'algorithme écrit à la question précédente est bon en supprimant la dernière valeur. Recalculer la médiane et la moyenne et conclure sur la stabilité des deux.

3.4 Transformée de Fourier

Récupérer le dossier `fourrier_analysis` sur la page web d'Éric Beucler. Compiler (*make makefile*), exécuter et regarder le signal (signal.asc) et le spectre (fort.1012) avec le logiciel `xmgrace`.
`>> xmgrace fort.1012`

Références

Delannoy, C. (1993). *Programmer en Fortran 90 (Guide complet)* (2003 - 5ème ed.). Eyrolles.

Lignelet, P. (1996). *Manuel complet du langage Fortran 90 et Fortran 95*. Dunod.